

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_l9i99y4g4pr9)

[**Table of Contents 2**](#_f7kpj8quu9aj)

[**Document Revision History 2**](#_jqqru6asuzpd)

[**Executive Summary 3**](#_414tddsvog9t)

[**Requirements 3**](#_i7jrdzxq2kv4)

[**Design Constraints 3**](#_bw19gfm5pwxb)

[**System Architecture View 3**](#_4q75gfylujnc)

[**Domain Model 3**](#_zdkbuumxq9fn)

[**Evaluation 4**](#_kstmdn372th)

[**Recommendations 5**](#_60wvyr4w0xh0)

## [Document Revision History](#_harzms6omi2a)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <03/23/25> | Shokhrukh Janobilov | <Brief description of changes in this revision> |

### Executive Summary

Creative Technology Solutions (CTS) is partnering with The Gaming Room to transform their Android game “Draw It or Lose It” into a web‐based, multi-platform application. The software design addresses the client’s key requirements, support for multiple teams and players with unique names, and ensuring only one instance of the game service exists at any time. Our solution leverages established object‑oriented principles and design patterns to achieve a robust, scalable, and secure application environment.

### Requirements

The application must:

* Support one or more teams per game.
* Allow multiple players per team.
* Enforce uniqueness of game, team, and player names to prevent duplicate entries.
* Guarantee that only one instance of the game service is active in memory at any time.
* Utilize design patterns (Singleton for the game service and Iterator for unique name checks) to streamline data management.
* Be developed in Java and deployable across multiple platforms including Linux, Mac, Windows, and mobile devices.

### Design Constraints

* **Web-Based Distributed Environment:** The application must support a distributed system where client devices (desktop and mobile) communicate securely with a centralized game server.
* **Performance:** Real-time drawing and rapid clue rendering require efficient memory and processing management.
* **Scalability:** The system should easily scale to support an increasing number of simultaneous games, teams, and players.
* **Security:** Protection of user data and prevention of duplicate naming must be ensured through secure coding practices and proper validation.
* **Platform Independence:** The design should consider the differences in development and hosting environments across Linux, Mac, Windows, and mobile devices.

### System Architecture View

The application adopts a multi-tier architecture:

* **Presentation Layer:** A responsive web interface that supports various devices and operating systems.
* **Business Logic Layer:** A singleton GameService managing the application’s core functions, including game, team, and player management.
* **Data Layer:** A mechanism (e.g., a relational database) that stores unique identifiers and entity data. Communication from frontend to backend is handled through RESTful services, ensuring efficient data exchange and secure connections.

### Domain Model

The domain model is built on an abstract base class, **Entity**, which encapsulates the common attributes id and name. This ensures consistency and code reuse across all entities in the system. The concrete classes – **Game**, **Team**, and **Player** – extend the Entity class. The **GameService** class manages collections of these entities. It uses the Iterator pattern to search through lists and enforce unique names, which fulfills the client’s requirement for uniqueness in game, team, and player entries.

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

## [Evaluation](#_5kozvt7121fh)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | macOS supports server software like Apache and Nginx, but it’s less commonly used in large-scale production. Limited vendor support and higher hardware costs may create issues. macOS is not typically chosen for scalable cloud infrastructure. | | Linux (  e.g., Ubuntu Server, CentOS) is the most widely used OS for web application hosting. It offers high scalability, open-source flexibility, stability, and robust community support. Licensing costs are typically $0. | | --- |  |  | | --- | | Windows Server supports IIS and a wide range of tools for enterprise deployments, but licensing can be expensive. Maintenance and support are excellent in enterprise settings, making it common for corporate environments. | Mobile devices are not designed for server hosting. However, they are the favorites for consuming the frontend. Backend hosting must be done on traditional server platforms (Linux, Windows, or cloud). |
| **Client Side** | The app must be designed as a responsive web application. macOS users typically access web apps through Safari or Chrome. Ensuring compatibility with Apple’s WebKit is essential. | | Linux clients typically use Firefox or Chromium-based browsers. A responsive HTML5-based design ensures cross-browser compatibility. Developers usually test layout behavior on various desktop environments. | | --- |  |  | | --- | | | Windows users access web apps via Chrome, Edge, or Firefox. Responsive design must support high-resolution displays and older hardware/software combinations. | | --- |  |  | | --- | | | The app must be fully responsive and optimized for both Android and iOS browsers (Chrome, Safari). Mobile compatibility testing is crucial, including touch interactions, screen size adaptation, and performance tuning. | | --- |  |  | | --- | |
| **Development Tools** | Xcode is mainly for native macOS/iOS apps. For web development, IntelliJ IDEA, VS Code, and Eclipse are popular. JavaScript/TypeScript, HTML5, CSS, and Java (Spring Boot) are common. Licensing costs are low or free. | | Linux developers often use VS Code, Vim, Eclipse, or IntelliJ. Most tools and SDKs are free. Full-stack JavaScript or Java with Spring Boot is typical. Git and Maven integrate smoothly in this environment. | | --- |  |  | | --- | | | Windows offers great IDE support (IntelliJ, VS Code, Eclipse, Visual Studio). While some enterprise tools carry licensing fees, most open-source stacks (like Node.js, Java) are supported well. | | --- |  |  | | --- | | Mobile developers use tools like Android Studio (for Android), Xcode (for iOS), or cross-platform frameworks like Flutter and React Native. Web-based UIs require responsive design testing with Chrome DevTools and other simulators. |

#### Recommendations

#### 

#### Operating Platform

For hosting *Draw It or Lose It* across different systems, I recommend using Linux, specifically something like Ubuntu Server. Linux is known for being fast, secure, and widely used in cloud environments like AWS and Google Cloud. It’s also free to use, which helps reduce costs for The Gaming Room. Since the game needs to handle lots of players at once, Linux is a great fit because of how well it manages performance and scalability.

#### Operating System Architectures

Linux uses a monolithic kernel design, which basically means it keeps most core functions in one place for better performance. Even though it’s “monolithic,” it’s still modular and allows flexibility through different components and add-ons. This helps when you need to customize or optimize parts of the system, like handling game sessions or user data. Linux also supports multitasking and strong process isolation, which is important when different players are interacting with the game at the same time.

#### Storage Management

Since the game relies on a large set of high-quality image files , we need a solid storage plan. A cloud-based storage system like Amazon AWS or Google Cloud Storage would work best. These systems are easy to scale and very reliable. They can store the images and serve them to users quickly no matter what platform they’re playing on. For user info and game data, a cloud-hosted relational database like PostgreSQL or MySQL would be great to keep things organized and easy to manage.

#### Memory Management

The game will need to load and display images really quickly, especially since each round only lasts a minute. Linux has some helpful memory management tools built in:

* It caches data and manages memory efficiently in the background.
* It can unload unused items from memory to make room for new stuff.
* Developers can also preload images or load them just-in-time to keep things smooth for the user.

These features help the game stay fast and responsive, especially when lots of users are playing at the same time.

#### Distributed Systems and Networks

Since players might be on different devices and platforms, the game will need to run in a distributed system—basically meaning the app runs on multiple servers and talks to different clients (like phones, laptops, etc.). To do this:

* The game backend should offer REST APIs that mobile and web clients can call.
* Use WebSockets for real-time communication during the game rounds.
* Load balancers can help spread out traffic and keep the game from crashing.
* Containers like Docker and tools like Kubernetes can help run the game in a way that scales up or down based on how many people are playing.

This setup will help the game handle more users and recover quickly from problems like server outages.

#### Security

Security is very important, especially since people will be playing across different platforms. Here’s how we can protect user data:

* Use HTTPS to encrypt all communication.
* Implement secure login systems like OAuth or JWT to manage sessions.
* Sanitize user input to avoid common web attacks like SQL injection or cross-site scripting.
* Use role-based permissions so only certain users (like admins) can access sensitive parts of the system.
* Cloud services also offer built-in identity and access controls that limit who can access what.

With these protections in place, players can trust that their information is safe, no matter what device they’re playing on.